42390P12319 PATENT

UNITED STATES PATENT APPLICATION

for

DYNAMIC COMPOSITION AND MAINTENANCE OF APPLICATIONS

Applicant:

Victor N. Vu

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Boulevard
Los Angeles, CA 90026-1026
(303) 740-1980

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: EV052961940US
Date of Deposit January 7, 2002
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Debbie Peloquin
(Typed or printed name of person mailing paper or fee)

Debbie Peloquin
(Signature of person mailing paper or fee)

DYNAMIC COMPOSITION AND MAINTENANCE OF APPLICATIONS

COPYRIGHT NOTICE

[0001] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software and data as described below and in the drawings hereto: Copyright © 2001, Intel Corporation, All Rights Reserved.

FIELD OF THE INVENTION

[0002] This invention relates to the field of application composition and maintenance, and, more specifically, to the dynamic composition of applications.

BACKGROUND OF THE INVENTION

[0003] Building a software application can be a very timely and tedious process. This process may consist of writing a technical specification, pseudocode, or any combination thereof. Thereafter, an application is written in a selected programming language, compiled, debugged, and executed. Modifying a software application can sometimes be just as tedious since a component must be recompiled every time it is modified.

[0004] The traditional software development process statically links needed components (functions, etc.) at link time. This means that any

components that are to be modified must be modified before an application that uses the component is executed at runtime. Consequently, if the modifications are being worked on, or are incorrect, the application must be shut down until modifications to the component are complete.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0006] FIG. 1 is a block diagram illustrating a system architecture for implementing a dynamically composed application in accordance with embodiments of the invention.

[0007] FIGS. 2A-2M contain sample code for the identification and versioning of components, lifetime management of components, communication capability of components, and event notification.

[0008] FIGS. 3A-3C contain sample code for the occurrence of events.

[0009] FIG. 4 contains sample code for component communication with external entities.

[0010] FIG. 5 is a block diagram illustrating the interaction between a component framework and components in accordance with embodiments of the invention.

[0011] FIG. 6 is a block diagram illustrating inter-component communication in accordance with embodiments of the invention.

[0012] FIG. 7 is a flowchart illustrating a method to dynamically compose

and maintain an application in accordance with general embodiments of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0013] In one aspect of the invention is a method to dynamically compose and maintain applications. An application is created within a component framework, where each component of the application is part of the component framework. Additionally, each component implements an integration interface. The integration interface comprises a number of methods that allow the component framework to manage the lifetime of the components, and that give the components the ability to communicate with the component framework, other components, and external entities.

[0014] The present invention includes various operations, which will be described below. The operations of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the operations. Alternatively, the operations may be performed by a combination of hardware and software.

[0015] The present invention may be provided as a computer program product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks,

CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs (Read Only Memories), RAMs (Random Access Memories), EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electromagnetic Erasable Programmable Read Only Memories), magnetic or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions.

[0016] Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection). Accordingly, herein, a carrier wave shall be regarded as comprising a machine-readable medium.

Introduction

[0017] FIG. 1 illustrates a system architecture for implementing a dynamically composed application. The architecture 100 comprises a component framework 102, one or more components 104A-N, a messaging mechanism 106, a component loader 108, an interface clearinghouse 110, and a components repository 112. Each application comprises a component framework and a set of components, and each component 104A-N is associated with a corresponding interface 114A-N.

[0018] Thus, when an application is created, it is created within a

component framework 102, where each of its components 104A-N implements an integration interface114A-N. Furthermore, an integration interface 114A-N comprises a number of methods that allow the component framework to manage the lifetime of its components. These are further described below.

Component

[0019] Components are atomic and comprise a set of constructs (i.e., classes in C++, functions in C, or software elements), and resources (interface for integration and communications). Atomic components must be deployable in a component framework: component A should not depend on a function in another component to be complete.

[0020] Components may be packaged using asymmetric encryption to prevent malicious tampering and to identify the makers of the components via their digital signature. The makers of the components use their private key to package the components. The makers of the components give the public key to the component framework at start-up time so that the component framework can unpack the packaged components and obtain the digital signature.

[0021] Each component 104A-N implements an integration interface 114A-N that allow its corresponding component framework 102 to manage it.

Component Framework

[0022] A component framework 102 provides a structural frame within

which the set of components 104A-N operate. Its purpose is multi-fold. For one, it is responsible for loading and unloading components 104A-N at runtime. When the component framework 102 receives a command to load and link a component 104A-N from an external entity 118, such as a GUI (graphical user interface) based administrative application, the component framework loads the specified component from a source (components repository 112, persistent storage, sockets, file system, etc.), and then links the loaded components.

Transitioning Components

[0023] The component framework 102 also manages the lifetime of components by transitioning the components 104A-N from the initialization state to the destroy state. When a component 104A-N implements the integration interface 114A-N, it inherits methods of the integration interface 114A-N that allow the component framework 102 to transition the component 104A-N from state to state.

[0024] FIG. 2A is an example structure having event handlers, CompFrameworkInterface that defines these states, and these states are illustrated in FIGS. 2C-2F. The event handlers corresponding to these states are summarized below:

[0025] Initialize (FIG. 2C, lines 8-20): the component framework 102 gives a component 104A-N a chance to prepare to operate. The component 104A-N is expected to react to this call by initializing its internal data structure and by

acquiring needed resources. When a running component 104A-N reacts to this state, it should be ready to perform normal operations.

[0026] Replace (FIG. 2C, line 22 - FIG. 2E, line 8): the component framework 102 notifies the to-be-replaced component 104A-N that it is being replaced. The replace event handler gives the component framework 102 the interface to be given to the new component 104A-N for the purpose of transferring states. When the component framework 102 completely transfers the states of the to-be-replaced component 104A-N to the new component 104A-N, the component framework 102 delivers a stop event to the to-be-replaced component 104A-N. The component framework 102 subsequently starts the new component 104A-N.

[0027] Run (FIG. 2E, line 10 - FIG. 2F line 3): the component framework 102 transitions a component 104A-N to this state when all resources are ready for the component 104A-N to use in normal operating mode.

[0028] Stop (FIG. 2F, lines 5-13): the component framework 102 gives the component 104A-N a chance to clean up or to release resources (i.e., memory, file descriptors, revoking published interfaces, etc.) before the component framework 102 destroys or unloads the component 104A-N so that the stability of the whole system is maintained. This event handler may return an error code (line 11) if the component 104A-N cannot continue normal operation.

[0029] As illustrated in FIG. 2B, the component framework 102 also

communicates with loaded components 104A-N via the integration interface 114A-N. The component framework 102 can identify the components 104A-N, including their name and version, for managing the components 104A-N as follows:

[0030] getName method (FIG. 2B, lines 8-11) is used to obtain the name of the loaded component 104A-N.

[0031] getVersion method (FIG. 2B, lines 13-16) is used to obtain the version of the loaded component.

[0032] The initialize, replace, run, stop, getName, and getVersion, members are initialized by a corresponding component 104A-N.

Inter-Component Communication

[0033] The component framework 102 provides a mechanism for components 104A-N to register interfaces that are used by other components 104A-N for inter-component communication. As illustrated in FIG. 2G-2J, components 104A-N use the following event handlers of their corresponding integration interface 114A-N for registering their interfaces for inter-component communication:

[0034] Publish method (FIG. 2H, line 4 - FIG. 2I, line 4) for the loaded component 104A-N to publish data and events.

[0035] Remove method (FIG. 2I, lines 6-20) allows a loaded component

104A-N to remove the previously published interfaces.

[0036] Retrieve method (FIG. 2I, line 22 - 2J, line 16) for the loaded component 104A-N to subscribe data and events.

Events

[0037] The component framework 102 may also inform registered components of the occurrence of certain events, such as when components 104A-N started or stopped, or when an interface is published or removed. As illustrated in FIG. 2K-2M, any components 104A-N that are interested in those events can register with the component framework to be notified of when those events occur.

[0038] addListener method (FIG. 2K, line 14 - FIG. 2L, line 7) allows a loaded component 104A-N to register an event handler or a listener for the specified event. When the specified event occurs, the given listener (event handler) is invoked.

[0039] removeListener method (FIG. 2L, line 9 - FIG. 2M, line 1) allows a loaded component 104A-N to de-register (remove) the previously registered event handlers or listeners.

[0040] These members are initialized by the component framework before the initialize method is called.

[0041] These events are illustrated in FIG. 3A-3C, and are summarized

below:

[0042] Component Started (FIG. 3A, line 3) event occurs when the component framework 102 starts a component 104A-N.

[0043] Component Stopped (FIG. 3A, line 4) event occurs when the component framework 102 stops or replaces a component 104A-N.

[0044] Interface Issued (FIG. 3A, line 5) event occurs when a component 104A-N publishes an interface.

[0045] Interface Removed (FIG. 3A, line 6) event occurs when a component 104A-N removes a previously published interface.

[0046] Command Issued (FIG. 3A, line 7) event occurs when a component 104A-N registers an administrative command.

Messaging Mechanism

[0047] Referring back to FIG. 1, the component framework 102 must support several levels of communication, including communication with external entities 118 so that various commands can be received. These commands include those affecting the lifetime of components 104A-N, such as the starting and stopping of components. Other commands, such as replacing a buggy with a good component 104A-N, shutting down the component framework 102, etc., are also supported.

[0048] A messaging mechanism 106 in embodiments of the invention provides the capability to listen to external entities 118 for commands. The mechanism, also known as dynamic commands facilitator (DCF) provides components 104A-N a way to dynamically register needed administrative commands and associated callback functions at runtime. When the DCF receives entered commands, the DCF forwards the entered commands to the components 104A-N that own the commands via the given callback functions. FIG. 4 illustrates definitions that external entities 118 can use to communicate with the component framework 102 and running components 104A-N.

[0049] The messaging mechanism 106 can be implemented via a message queue, sockets, or any other IPC (Interprocess Communication) mechanism.

Component Loader

[0050] A component loader 108 is responsible for loading the specified components 104A-N from a source such as a file system or a components repository 112 (see below). After the specified components 104A-N are loaded, they are linked into the running application. The component loader 108 should be capable of unpacking a packaged component 104A-N using the public key, which is given by the user at the start-up of the component framework 102. The component loader 108 should also be capable of loading packaged components 104A-N from the specified location, such as permanent storage, or a network

connection.

[0051] The component loader can also contact a trusted central host in a network to query and to retrieve components. In addition, if the component loader is configured with references to its peers, such as other component loaders in a network, the component loader will query its peers and retrieve the needed components from the peers who have the fastest response time. If network connection problems occur during transferring of components, the component loader will try to retrieve the needed components from other peers regardless of the response time.

Component Repository

[0052] The component repository 112 holds the packaged components 104A-N. When the component loader 108 needs to load a component 104A-N, the component loader 108 loads the packaged component 104A-N using the given public key to decrypt the packaged component 104A-N and to verify the digital signature of the maker of the component 104A-N in effect.

Interface Clearinghouse

[0053] The component framework 102 should provide a well-known place to collect, classify and distribute interfaces. This place is known as the interface clearinghouse 110. The purpose of the interface clearinghouse 110 is to provide a mechanism that the components 104A-N can use to publish and remove

interfaces.

[0054] If a component 104A-N wishes to share its internal data and events with other components 104A-N, it publishes its interfaces to the interface clearinghouse 110. The components 104A-N that wish to consume the published data and events come to the interface clearinghouse 110 to retrieve to the published interfaces.

[0055] The interface clearinghouse 110 should not impose any wire protocols. The components 104A-N agree with each other on suitable wire protocols, giving the components 104A-N the flexibility in creating new and suitable wire protocols.

Communication Between Component Framework and Components

[0056] The component framework 102 for a given application loads and links a specified component 104A-N when the application receives a start command. The application first locates the specified shared library that contains the component 104A-N, using, for instance, the following command:

[0057] Start <Component-Package-Name> <Interface-Accessor>

[0058] Component-Package-Name is the name of the encrypted package that contains the component 104A-N and associated resources. Interface-Accessor is the function name that is used by the component framework to retrieve an integration interface from the component 104A-N being loaded.

When that function name is invoked by the component framework 102, that function must return an integration interface 114A-N. The external entity 118 that starts that component 104A-N must specify that function name (Interface-Accessor) so that the component framework can initiate a communication session to retrieve an integration interface from the component.

[0059] The component framework 102 loads the found shared library comprising the specified component 104A-N and then retrieves the integration interface 114A-N via an interface-accessor as specified by the external entity that starts a component. For example:

[0060] void* <interface-accessor-name>(void) {...}

[0061] The interface accessor then returns a pointer to an instance of the integration interface114A-N. The pointer stays valid throughout the life of the component 104A-N.

[0062] The component framework 102 can then use the retrieved integration interface 114A-N to transition the loaded and linked component 104A-N to the initialize state by invoking the initialization method, and transitioning the linked component 104A-N to the run state by invoking the run method. If the running component 104A-N wishes to publish data and events for other components 104A-N to retrieve, the running component 104A-N invokes its publish method. If the component wishes to consume the published data and events, it invokes its retrieve method. If a running component wishes to receive

events, the running component invokes its addListener method.

[0063] The component framework 102 unloads the specified component 104A-N when the application receives a stop command for the component 104A-N of a given name and version. Upon receiving the stop command, the application queries all components 104A-N for their name and version. The first component 104A-N that has the specified component name and version is terminated. The component framework 102 gives the specified component 104A-N an opportunity to perform house cleaning by transitioning that component 104A-N to the stop state.

[0064] The application then exits when it receives an exit command. The application terminates all components 104A-N before it exits, in the same manner as if the application receives a stop command for all components104A-N.

[0065] FIG. 5 illustrates the interaction between the component framework 102 and the set of components104A-N. When a component 104A-N is loaded by the component framework 102, the component framework 102 invokes the initialize method 502 of the loaded component. When a running component 104A-N needs to be unloaded from the component framework 102, the component framework 102 invokes the stop method 504 of the loaded component 104A-N.

[0066] If the running component 104A-N wishes to publish data and

events for other components 104A-N to subscribe to, the running component invokes the publish method 506 of the loaded component 104A-N. If a component 104A-N wishes to consume published data and events, it invokes the retrieve method 508 of the loaded component 104A-N.

Communication Between Components

[0067] As illustrated in FIG. 6, after at least one call to both the publish and retrieve methods, a communication bus 600 for inter-component communication is established, allowing components 104A-N to communicate with each other without intervention from the component framework 102.

[0068] The components 104A-N themselves define a suitable wire protocol and interface. Components 104A-N use an interface to talk to each other, and the wire protocol is the language used to describe the intention of the components 104A-N.

[0069] The components 104A-N that produce data and/or events publish an interface via the interface clearinghouse 110 of the component framework 102. The components 104A-N that consume data and/or events subscribe to those data and/or events via the interface clearinghouse 110. As stated above, when there exists at least one call to both publish and retrieve, a communication bus 600 is established.

[0070] When a component 104A-N wishes to communicate with one or

more other components 104A-N, the consumer component invokes the retrieve method 508 of the integration interface 114A-N specifying an interface of the component 104A-N. The component framework 102 gives the specified interface to the component 104A-N if the requested interface exists. The existence of an interface only occurs when that interface has been published by a component 104A-N. The consumer component 104A-N uses the addListener method to notify the producer component 104A-N of the interested events.

[0071] Communications between components 104A-N can be based on events so that traffic is minimized. The components only receive the specified events from their peers and the component framework. For example, when an event occurs within a particular component and no other components register to receive that event, the occurred event will not be broadcasted. This constraint applies to all events and components. The components 104A-N must notify each other when any of the components 104A-N want to terminate a communication session. Furthermore, components 104A-N should be able to sense the abnormal termination of a communication session and react to it accordingly.

Rapid Composition of Applications

[0072] Using the described methods above, an application is capable of loading and linking a component at runtime. For example, assume an application comprises component A, B, C, and D, and the application is executing without component B because component B is being upgraded. Assume also that

component B is packaged in Component-Package-B. When component B is ready, it can be loaded and linked to the other components as described below.

[0073] The application receives a start command from an external entity, which provides the application with the interface accessor function, IA_B. The component framework for the application loads and links a specified component. The application first locates the specified shared library that contains the component, using, for instance, the following command:

[0074] Start <Component-Package-B > <IA_B>

[0075] The component framework loads the found shared library comprising the specified component and then uses the interface accessor function to retrieve the integration interface by invoking, for example, void* <IA_B >(void) {...}. The interface accessor then returns a pointer to an instance of the integration interface.

[0076] The component framework can then use the retrieved integration interface to transition the loaded and linked Component B to the initialize state by invoking the initialization method. Component B can then be utilized by the application.

Dynamic Maintenance of Applications

[0077] A defective and/or an obsolete component can be replaced with a new component at runtime. The user (administrator, technical support staff, etc.)

issues a replace component command to the component framework. The component framework notifies the running component of the replacement intention via the replace method of the integration interface. The replace method returns the state of the running component to be transferred to the new component. When the transferring of state is complete, the component framework stops the running component and starts the new component.

[0078] The user starts a diagnostic component. The component examines the state of all other running components. If any component is found to be not in a stable state, the diagnostic component issues a command to replace that faulty component. If an updated component is not found in the local component repository, the diagnostic component can contact a trusted central host to query and to retrieve an updated component to replace the faulty component. The retrieved updated component package shall be stored in the local component repository.

[0079] For example, an application that is capable of dynamically loading and linking components uses a load balancing algorithm. Suppose, however, that a user wants to use his own load balancing algorithm. A component comprising the user's load balancing algorithm is developed, for instance, in the C language.

[0080] The component is then compiled into a shared library. That shared library is then packaged using a private key whose associated public key is

already given to the component framework. A command is then issued to the component framework to load and link the component to the application. The application then delegates the load balancing task to the dynamically loaded and linked component. This delegation is made until the application is issued a command instructing it to use the original load balancing algorithm.

Method

[0081] FIG. 7 is a flowchart illustrating a method for dynamically composing and maintaining an application in accordance with embodiments of the invention. The flowchart begins at block 700 and continues to block 702 where an indication to integrate a component into an executing application is received. At block 704, the component is loaded, and at block 706, the component is linked by obtaining the component's integration interface and invoking the initialize method of the integration interface. The method ends at block 706.

Conclusion

[0082] Thus, an invention has been described for composing or maintaining an application at runtime without shutting down the running component framework.

[0083] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that

various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.